White Paper on the Next-Generation Data-Access Architecture for Naval C^{4}I Systems

Dr. Marion G. Ceruti
Space and Naval Warfare Systems Center, San Diego, CA

and

Mr. Scott A. Gessay
FGM, Inc., 131 Elden Street, Suite 108, Herndon, VA

ABSTRACT

The Joint Maritime Command Information System (JMCIS), is the primary Command, Control, Communications, Computers, and Intelligence (C^{4}I) system for the maritime services. To promote efficient information access in this system, the JMCIS Data Engineering Group formed a committee with a three-fold mission: to evaluated the state of new data-access technology, methods and architectures; to explore the systems and software developed within the C^{4}I sector that can use this technology; and to recommend an architecture on which to base the framework of data-access methodologies into which Naval C^{4}I systems could evolve. The various technologies and approaches available for the next-generation data-access methodology in Naval C^{4}I were investigated, including an examination of both commercial-off-the-shelf and government-off-the-shelf technologies, international standards, and general industry trends. The approach was to define the requirements, evaluate the available technologies, and to compare them to one another. The option of combining technologies was explored. The findings of this investigation are presented here.

I. INTRODUCTION

The Joint Maritime Command Information System (JMCIS) has been installed on ships and shore-based locations to meet the command, control, communications, computers, and intelligence (C^{4}I) requirements of the Navy, Marine Corps and the Coast Guard. (See, for example, [3].) The JMCIS plan calls for a common hardware and software to be installed on more than 200 ships [17]. The JMCIS database architecture is a critical part of the design of this system. Therefore, JMCIS Data-Access Committee (DAC) was established to provide support to the

Proceedings of the Fifteenth Annual Federal Database Colloquium, "Combining Emerging Technologies for the Information Systems of the Future," San Diego, CA, September 9 - 11, 1998.

JMCIS Data Engineering Group (DEG). Its mission was to accomplish three goals:

- Evaluate the state of technology concerning new data-access methods and architectures.
- Explore the systems and software developed within the C^{4}I sector that used this technology.
- Recommend an architecture on which to base the framework of data-access methodologies into which Naval C^{4}I systems could evolve.

The JMCIS DAC completed an 18-month investigation of various technologies and approaches available for the next-generation data-access methodology in Naval C^{4}I. This effort included the examination of both Commercial off-the-shelf (COTS) and Government off-the-shelf (GOTS) technologies, ANSI ISO standards, and general industry trends. (See, for example, [22].) The approach for this examination was to define requirements (both immediate and long term), evaluate the available technologies, and to compare them to one another. In so doing, the option of combining technologies instead of recommending a single technology or approach also was explored.

II. REQUIREMENTS

Standard Access Methods

The next-generation data-access architecture will need to provide several methods of information access for user applications. This section describes three fundamental access methods that will be required in this architecture.

Most importantly, a generic, standard access method is required for applications to interface to databases without being tied to one particular source of data or COTS-specific interface. This methodology should be robust enough to utilize all the advanced features of the target Database Management System(s) (DBMSs) and of the databases they manage, while preserving a standard interface method in the code of the developed client applications. This will ensure lower code maintenance costs, and the ability of the client applications to access a variety of data repositories with no code changes.

Secondly, the access method must include an object and Structured Query Language (SQL) "pass-through" capability, to allow a "native" connection directly to the data repository as though in a client-server architecture. (See, for example, [15, 18].) This is important for two reasons:

- To provide a flexible environment during migration to this new architecture, and
- To provide specific applications that cannot change or those that employ non-standard, specialized, advanced database features to continue to connect to the required data repositories.

Third, the next-generation data-access architecture will need to support access from web browsers. Web pages will be built with back-end interfaces directly hooked to data repositories. This technology, which is prevalent in the industry, will alleviate the need to replicate large amounts of data throughout C4I systems. Information access will be much faster for users when a "need-to-know" has been established. (See, for example, [4, 10, 16].)

Non-RDBMS or Lowered-RDBMS Dependence

JMCIS component systems are implemented with Oracle or Sybase DBMS servers. Informix DBMS is used by the U. S. Coast Guard and also by the systems that provide meteorological and oceanographic data to JMCIS. Client applications developed for JMCIS Ashore are written to interface with an Oracle server, whereas JMCIS Afloat variant client applications use a Sybase server. Therefore, to introduce JMCIS Ashore variant applications into a JMCIS Afloat system an Oracle server must be installed or the applications have to be rewritten to interface with Sybase. Also, JMCIS needs to interface with other DBMS server products, such as Informix DBMS.

The requirements in this area for the JMCIS data-access architecture are as follows:

- Provide the interfaces to reduce the application software's dependence on a specific vendor's RDBMS product.
- Application software must be portable to other RDBMS products.
- Reduce RDBMS dependence without settling on the least common denominator of functionality.
- Base the JMCIS data-access architecture interfaces on industry standards. Reduce RDBMS dependence using industry-standard interfaces, such as SQL Call Level Interface (SQL/CLI) and Remote Data Access (RDA), which are ANSI/ISO standards. (See, for example, [14, 15, 18].)
- Although complete RDBMS independence probably is not possible, reduce and manage RDBMS dependence without introducing a proprietary middleware or Fourth-Generation Language (4GL) dependence.
- Provide a framework for developing applications that can be ported efficiently to other RDBMS products.

Accommodation of New Database Technologies

Many new advanced database features that could be of great use to C4I systems are offered by RDBMS vendors. Several of these technologies are described below. Most importantly, many of these technologies can improve performance and interoperability significantly in C4I information systems. Therefore, it is essential to ensure that any new data architecture will be flexible enough to allow the use of these technologies where applicable within C4I. Accommodating these technologies is related to the concept of "pass-through" mentioned above. Proper architectural

planning will ensure that COTS expenditures on new technologies provide the maximum return on the investment.

Several technologies are available in the form of COTS products. One such technology is an HTML-to-RDBMS interface. This technology could streamline significantly custom-application development, while providing access to users in a fraction of the time and cost. Similarly, advanced replication techniques available from the major RDBMS companies is another mature technology that could solve a plethora of problems in C^4I. Thirdly, a COTS On-Line Analytical Processing (OLAP) and Decision Support Systems (DSS) technologies could solve many of the spatial data and Very Large Database (VLDB) issues in C^4I.

Data warehousing and data mining are becoming extremely useful data storage and application architectures that give users the ability to keep vast amounts of data on line. Data can be viewed from different perspectives to determine trend analysis, post-event analysis, etc.

Performance Issues

Database performance is a critical aspect of the JMCIS data-access architecture. The architecture must address query-response time, transaction processing, and concurrence. A major concern in designing a data-access architecture is the tradeoff between performance and portability (i.e. lowered RDBMS dependence). Typically, an interface layer is designed to provide for portability but this additional layer will degrade performance. However, if designed correctly the interface layer should provide for lowered RDBMS dependence with minimal impact on performance.

The following are performance requirements for the JMCIS data-access architecture:

- Support a three-tiered architecture consisting of clients, application servers, and resource (e. g. database servers). (See, for example, [6].)
- Support the concept of a database transaction as a unit of work.
- Maintain a database connection throughout the session.
- Support native RDBMS locking capabilities, such as row-level locking to provide the optimum transaction throughput and to minimize deadlocks.
- Support threads. Multithreading allows the system to split an application program to perform multiple tasks in parallel.
- Support load balancing.
- Support RDBMS parallel processing environments.
- Minimize network overhead through capabilities such as stored procedures and array processing.

Distributed, Heterogeneous Federated Database Architectures

The term, Federated-Database System (FDBS), was first defined to mean a collection of independent, pre-existing databases (for which the data administrators and/or the database administrators) agreed to cooperate [13, 14]. Thus, the database administrator (DA) for each component database provides the federation with a schema representing the data from his or her component that can be shared with other members of the federation [13]. In a broader architectural sense, an FDBS is a collection of cooperating but autonomous component database systems that are possibly heterogeneous [23]. The collection of databases that supports JMCIS fits this description. (See, for example, [5, 23].)

The next-generation, data-access system for JMCIS will include services designed to provide seamless access to heterogeneous, distributed data for applications and operational users. The technical approach will be based on general principles of federated-database management implemented with an open architecture that facilitates the integration of new segments, systems, and technology into the existing configuration, as they become available. Requirements are as follows:

- Hide the heterogeneity on the network from applications to provide a seamless, client-server interface to the data.
- Utilize intelligent, object-model and legacy-data-schema managers. (See, for example [6, 8, 9].)
- Utilize intelligent-query caching and auto-refresh capabilities.
- Dynamically identify changes in class and in data-source structure.
- Perform heterogeneous joins across multiple servers for ad hoc queries and applications that require them.
- Maintain relational reliability and flexibility.
- Implement a client-server architecture.
- Preserve the vast majority of the important, present capabilities visible to the user that are necessary for the complete function of current applications. (See, for example [21].)

The architecture and infrastructure will enable the system to evolve to satisfy the following long-term requirements:

- Introduce more object-oriented architectures, software and techniques into the JMCIS environment, including object-oriented data modeling, and other technology that will address the fundamental connection between the relational and the object-oriented models [6].
- Function according to approved procedures in an SCI environment using SCI databases, and operate on the JMCIS SCI LAN.
- Construct the data-access system using the tightly coupled federated database approach in which component databases offer shared data represented to the applications and users by a global schema.
- Provide the capability to perform joins at the federated database management level or "middle tier," thus relieving the applications level

of this task.

- Identify and resolve multiple levels of heterogeneity and the problems they introduce in the integration of legacy systems by instantiating objects with conflicting-data attributes from multiple sources.
- Optimize global query execution within a distributed object environment, thus addressing the performance implications of the additional software layers to minimize their impact on the efficiency with which JMCIS applications can obtain their required data.
- Sense the network configuration and report to the user the nodes that are connected actively to the network.
- Require little or no recoding of existing applications and no redesign of legacy databases, as a minimum for participating in the federation.
- Provide access to multi-media databases containing graphic, audio, video, binary, and multidimensional data objects that may reside in flat-file, relational or object-oriented DBMS formats.
- Use GOTS components where possible to obtain maximum leverage from funding of other sponsors. Keeping the cost down is a high priority.
- Minimize the requirement of sites to purchase additional software licenses by including the functionality of COTS middleware products.
- Provide a smooth transition to migrate data access from the present methods to the more-advanced information services and technologies.

Objects and Object-Oriented Database Management System (OODBMS)

The object-technology requirements for the data-access system are as follows:

- Provide a database-processing capability for both the relational and object-oriented approaches to C^4I System Development.
- Provide Object-Relational technology that results in data-source independence.
- Provides an architecture that allows DoD to "plug and play" data management systems.
- Optimize global query execution within a distributed-object environment.
- Notify client automatically of changes to registered objects of interest.
- Perform object-oriented data modeling.
- Provide interfaces that fully comply with the Common Object Request Broker Architecture (CORBA).
- Offer object-query relaxation techniques.
- Offer object-query optimization techniques.
- Support fine-grain client C++ proxy objects. (This is a performance issue also. Server objects are instatiated on the client platform to minimize network traffic and the consequent performance degradation.)

The architecture and infrastructure will enable the system to evolve to satisfy the following long-range, object-oriented requirements:

- Introduce more object-oriented architectures, software and techniques into the JMCIS environment, including object-oriented data modeling, and

other technology that will address correctly the fundamental connection between the relational and the object-oriented models.

- Utilize not only fully CORBA-compliant interfaces, but also demonstrate portability between the different CORBA instantiations, so that an ORBIX-developed CORBA application could interface to a different CORBA application without being redesigned or recoded.
- Identify and resolve multiple levels of heterogeneity and the problems they introduce that are inherent in the integration of legacy systems by instantiating objects with conflicting data attributes from multiple sources.
- Provide for the establishment of intelligent, object-model and legacy -data-schema managers.
- Implement intelligent placement and replication of data objects based on access considerations, including but not limited to priority and frequency
- provide data-source transparency to the user.
- Dynamically identify class and data-source structural changes, and incorporate them into the metadata schema automatically.
- Provide clients with automatic notification of changes to registered objects of interest.
- Include multiple options for Application-Program Interfaces (APIs), including full C++ language integration, standard C code, and ORBIX.

Redundant and Prioritized Database Access

Requirements for the near term are as follows:

- Provide data access methods to selected JMCIS data sources.
- Provide read-only or read-write, data-access capabilities for segments and applications, depending on their purpose.
- Use redundant data sources automatically for first-level fault tolerance to provide data-source transparency to the user.
- Provide intelligent placement and replication of data objects based on access considerations, including but not limited to priority and frequency.
- Provide synchronous, asynchronous and query scheduling capabilities.

The architecture and infrastructure will allow the system to evolve to satisfy the following long-range requirements:

- Make available to the user and applications (consistent with user profiles) the capability to read, write, update and delete both static and dynamic data, including data from automatic message handling, depending on the purpose of the various segments and applications.
- Provide a flexible, incremental development environment to accommodate the next industry or Government standard, whatever that may be.
- Provide a conceptually shared and global JMCIS database that will be physically distributed and somewhat redundant, (to avoid a single point of failure.)

Interoperability with GCCS

The Global Command and Control System (GCCS) provides a Common Operating Environment (COE) that defines a set of core services for mission applications [7, 20]. Data-access services are a component of the GCCS-COE architecture. Navy-mission applications will have to operate within the GCCS COE.

The following sections identify the requirements for GCCS interoperability in the JMCIS data-access architecture.

- The JMCIS data-access architecture must be able to operate within the GCCS Common Operating Environment (COE).
- The JMCIS data-access architecture must be able to operate within the GCCS database server's runtime environment.
- The JMCIS data-access architecture must comply with the GCCS database server's integration standards.
- The client applications of the JMCIS database must be able to operate within the GCCS COE.

Ensuring an Efficient Migration Path

JMCIS has many applications that interface with either the Ashore or the Afloat server. These interfaces vary from vendor APIs to embedded SQL. Applications using these interfaces must migrate to the JMCIS data-access architecture.

Software and databases can exist at various levels of compliance with a standard, common operating environment. The point at which one observes software module or data segment on the migration path will be determined primarily by what has been done to comply with these standards. For example, a migration database that has undergone little or no modification may be at an "entry level" in an FDBS, whereas, a database system that was designed specifically with a particular federation in mind may be fully compliant with the rules of the federation as soon as it is integrated. Most JMCIS software and databases fall somewhere between these two extremes.

The following sections identify the requirements for ensuring the migration of legacy applications in the JMCIS data-access architecture. In particular, the data-access architecture should provide the following capabilities:

- Tools and/or libraries for phasing legacy applications into the architecture.
- A migration path with multiple levels of compliance that allow applications and systems to phase into the overall architecture.

Emerging Standards

When defining the next-generation data-access architecture, careful consideration should be giving to trends in industry with respect to ANSI and ISO standards. In the past, focus has been placed on building C^4I database applications according to what was standardized already. This has put many C^4I applications at a disadvantage because some or many advanced, database technologies have gone unused, due to the lack of accepted standards at the time of development. Many of these technologies (such as triggers, replication, etc.) could have been used by keeping abreast of ongoing standardization efforts, so as not to "go out on a limb" technologically. For the next-generation architecture, it is important to embrace these emerging standards and utilize the advanced technologies provided by COTS DBMS products that are destined to become standards. More on this topic can be found in the section on Current Access Methods, Technological Advancements, and Emerging Standards.

III. THE CURRENT STATE OF TECHNOLOGY AND CASE STUDIES

ORB Architecture - The JTF-ATD Data Server

The Joint Task Force Advanced Technology Demonstration (JTF-ATD) Data Server project is sponsored by the Defense Advanced Research Projects Agency (DARPA). The JTF-ATD Data Server's advanced information services can provide the required client-server communications between applications and the DBMSs throughout the JMCIS network. This will eliminate the need for DoD to continue to purchase additional COTS, client software for the database manager. These services will also provide capabilities for minimizing the effort to rehost JMCIS segments in this advanced architecture. This can be done by exploring the approach of providing libraries that interface with the OODBMS services and that mimic the C-library interfaces of the Oracle, Sybase, Informix, and other RDBMSs.

The JTF-ATD Data Server, which has played an important role in the Joint Warrior Interoperability Demonstration (JWID), is considered one of the highlights of the very-successful technologies that were demonstrated. The JTF-ATD Data Server has received a great deal of attention in industry, government and academia.

The JTF-ATD Data Server complies with CORBA, which consists primarily of the object model, the Object Request Broker (ORB) and object adapters, and the Interface Definition Language (CORBA-IDL). Each component is discussed below. The information on CORBA discussed here is from [6] and [19]. (For further information on object technology, see also [8].)

The object model describes object semantics and object implementation. Object semantics include the semantics of an object, type, requests, object creation and destruction, interfaces, operations, and attributes. Object implementation

includes the execution model and the construction model. In general, the essential constructs of most object models can be found in the object model of CORBA.

An essential feature of the ORB is that it enables communication between a client and a server object. A client invokes an operation on the object and the object implementation consists of the code and data needed to implement the object [19]. The ORB provides the required mechanisms to identify the object implementation for a particular request and enables the object implementation to receive the request. The ORB also provides the communication mechanisms needed to deliver the request. Furthermore, the ORB supports the activation and deactivation of objects and their implementations. The ORB generates and interprets object references. To summarize, the ORB provides the mechanisms to locate the object and communicate the client's request to the object. The client does not need to know the exact location of the object or the details of its implementation. Objects use object adapters to access the services that the ORB provides.

IDL is a declarative language that describes the interfaces that the object implementations provide and that the client objects call. It should be noted that the clients and object implementations are not written in IDL. The IDL grammar is a subset of ANSI C++ with additional constructs to support the operation invocation mechanism. An IDL binding to the C language has been specified, whereas other language bindings are in progress. IDL is used to communicate between a client and a server in the following manner. Two types of modules, the IDL stub and the IDL skeleton, are connected to the ORB core. The client's request is passed to the ORB core via an IDL stub, and an IDL skeleton delivers the request to the server object from the ORB core.

CORBA can be used for Integrating Heterogeneous Database Systems. Some directions on using CORBA for this purpose are described below.

A major motivation for adopting a CORBA-like approach to the integration of heterogeneous databases is the complexity of migrating legacy databases to new-generation architectures. Whereas the migration of such databases and applications to the client-server architectures is desirable, the costs of such migration can be enormous. Therefore, a better approach is to keep the legacy databases and applications and develop mechanisms to integrate them with new systems. These mechanisms include the approach of the distributed-object-management systems in general and the CORBA approach in particular.

The major advantage of the CORBA approach is the ability to encapsulate legacy database systems and databases as objects while eliminating the need for major modifications [6]. However, the techniques to handle the various types of heterogeneity are still needed. This is because CORBA itself does not handle some problems like transaction heterogeneity and semantic heterogeneity. However, the procedures for handling the various types of heterogeneity can be encapsulated in

the CORBA environment and invoked appropriately. These concepts are illustrated below with some examples.

Consider the need for clients to communicate with a group of database servers. One way is to encapsulate the database servers as objects and have the clients issue appropriate requests and access the servers through an ORB. If the SQL-based servers are used, the entire SQL query or update request could be embedded in the message. When the method associated with the server object gets the message, the method can extract the SQL request and pass it to the server for execution. The results from the server objects are encoded as a message and passed back to the client via the ORB.

Next, consider the issue of how to deal with a particular type of heterogeneity. Suppose a SQL-based client is present with a server is some legacy database system based on the network model. In that case, the client's SQL query will need to be transformed into an appropriate language that the server can understand. (For more information on the issues of transforming one representation scheme into another, see [6].) The client's request is sent to the module responsible for performing the transformations. This module, called the "transformer," could be encapsulated as an object. The client's SQL request is sent to the transformer, which converts the request into a format that the server can understand. The transformed request is sent to the server object. Note that the transformer could transform the SQL representation directly into a network representation or it could use an intermediate representation to complete the transformation.

The distributed processor, which is a module that can perform the functions of distributed-data management, is responsible for handling functions such as global- query optimization, and global transaction management. This module also can be encapsulated as an object and processes the global requests and responses. The server assembles the response sent to the transformer to convert into a representation that the client can understand. All the communications are carried out through the ORB [6].

Distributed Computing Environment (DCE) Architecture

DCE is a technology that could evolve into an industry standard method to provide distributed computing access. In addition to DCE's many positive technical features and services, the GCCS community has considerable interest in DCE. Moreover, DCE can provide support to JMCIS as well. (See, for example, [1, 2].)

The Open Group's DCE provides a set of services that address the problems found in distributed client-server environments today. The core components are remote procedure calls (RPCs), directory services, and security services. The goal of DCE is to provide solutions for the problems inherent in distributed computing and to make sure those solutions work well in a complex, multi-vendor world.

The DCE RPC models two distributed processes as a subroutine and a caller of the subroutine. The fact that these processes may run on different machines connected by a network is hidden from the programmer. The client and server can communicate, locate one another on the network, and convert data formats via RPCs. One of the reasons so many developers are interested in working with the DCE RPC is that it functions independent of any particular protocol or network type.

With many systems on a network, providing clients with the ability to locate servers is important. Using two DCE components, the Global Directory Service (GDS) and Cell Directory Service (CDS), a hierarchy is produced in which the names and attributes of systems are supplied throughout the network. GDS and CDS provide ways for applications to locate one another. These services provide a way for the servers to store information that clients will need to contact those servers. GDS and CDS also provide those clients with a way to retrieve the information.

One of the more critical components of DCE is the security service. DCE provides four key security services: authentication, authorization, data integrity, and data privacy. DCE supports authorization using POSIX-based, access-control lists (ACLs). Because distributed security requires clock synchronization among machines, DCE provides a Distributed Time Service (DTS) that performs this function.

Two other services that DCE supports are threads and a Distributed File System (DFS). Based on standard, POSIX interfaces, threads provide a means to improve application performance by implementing parallelism. DFS allows a program to access files on the file server just as though they were located on the local system's disk. This goes beyond an ordinary network-file system (NFS) because with DFS every node in the network identifies the same file by the same name and sees it located in the same directory thereby hiding the physical layout of the network. Whereas DFS is officially part of DCE, DFS is really an optional application built on top of the core components.

DCE provides the foundation and some tools that allow applications to interface with different operating systems, network protocols and databases in a distributed environment. DCE has a set of APIs that developers can use to build client-server applications. Various third-party vendors have built additional layers on top of DCE to support database client applications, which could represent a cost savings if implemented throughout JMCIS. Without this third-party product, a programmer would have to become familiar with the 600 DCE API calls and would need to know about multithreaded applications, the ACL manager, and server initialization.

A second generation of client-server computing lends itself well to the DCE environment. A three-tier architecture that uses a client workstation, application server and a database server is a trend that benefits the higher-end applications. With the application server centered in a three-tier architecture, organizations can

achieve higher availability and performance, along with the benefits of transaction-processing (TP) monitors. The benefits of a TP monitor include:

- control of a single unit of work through a two-phase commit protocol against distributed, heterogeneous databases, and
- high availability and performance because a TP monitor can use multiple regions to balance the workload.

Disadvantages include:

- higher complexity in coding for TP monitors, and
- the need for the application developers to code explicitly many of the features that the monitor supports.

The most popular way of accessing databases in the DCE environment appears to be through middleware from third-party companies, especially those that have formed alliances with the major DBMS vendors, such as Oracle Corp.

The more a heterogeneous environment is characterized by marked dissimilarities between the components, the more beneficial DCE will be in that environment. DCE satisfies many of the core interoperability, security and transparency requirements needed for an open distributed processing solution. Whereas this may be true for applications to interact and security to be maintained, when the differences between database components are extreme, access becomes very difficult especially if the software has been developed with DCE APIs. The support of multiple databases by third party vendors needs periodic investigation.

One major benefit of DCE is the security service. If this can be exploited through APIs or third party vendors when writing database applications, data access in the DCE environment may have an advantage over a traditional client-server architecture. (For more information on database security, see [11].)

How does a DCE-based application compare in performance to the identical application written with sockets? DCE will be less efficient than a simple TCP-based connection. This performance issue will also be somewhat of a problem for data access in the DCE environment versus in the typical client-server architecture. However, traditional applications won't be as robust or provide DCE services. Without middleware to provide seamless access to databases, programmers must understand RPC, security and networking, as well as have threads experience. This is considered one of the biggest disadvantages to creating client/server applications within DCE.

In summary, the attractive features of DCE can provide some needed services for the distributed environment of JMCIS. Some of the problems with DCE may be solved by combining DCE with other software. Whereas DCE alone is not a complete solution, it would add utility and flexibility to the data-access approach in JMCIS.

COTS-Specific, Middleware-Enabled Architecture

Advancements in the area of middleware have provided the industry with significantly enhanced capabilities with regard to client applications accessing various data repositories. Companies such as Sybase and Oracle have produced very robust midleware products. One such middleware product, *OmniSQL Gateway*, is a part of the *Sybase Enterprise Connect* product line. For the purposes of discussion, the OmniSQL Gateway product was chosen as an example to demonstrate the range of possibilities with middleware technology.

The OmniSQL Gateway was one of the first RDBMS middleware products released in the industry, and offers some very interesting capabilities of dealing with a federated, heterogeneous database schema. This section explains how this product might be used in the next-generation data-access architecture for Naval C^4I.

The entire framework of today's RDBMS middleware products revolves around the concept of a "database broker" that acts as gateway to link user applications to required data repositories. Applications must have been written using embedded SQL, and must not stray too deeply into the proprietary SQL dialect of the original RDBMS, or else the gateway will be unable to translate to a different data repository (thus requiring a "pass-through" method of data access). This is the main limitation of current middleware products, pending the standardization of the various components concerning data access. Metadata are stored in the middleware product; in OmniSQL Gateway, this information actually resides in local database tables. When applications require access to data repositories, lookups in Omni's local tables are enabled to determine the database to access, how to deal with the interface, how to construct the SQL statement (not done in pass-through mode), and how to complete the transaction.

Data sources are accessed via the gateway. This requires preparation and maintenance of the metadata tables in the gateway itself. Translation occurs to some level of the native SQL dialect of the target data repository. The OmniSQL Gateway also provides a "pass-through" capability, for access to data repositories preserving the original SQL dialect to the target DBMS.

By design, Sybase and other middleware vendors would prefer that their particular access protocol and bindings be used from all client applications to the middleware gateway. For the Sybase OmniSQL Gateway, the access would be through *Open Client* to *Open Server*. The advantage is that customers can standardize to a single access method, and let the gateway deal with the translational issues. Another benefit is gained in that a prioritization schema can be employed for redundant data repository access. Significant flexibility is achieved using this approach, and the Navy's initial effort testing the OmniSQL Gateway in Philadelphia in 1994 showed that performance issues were not a big concern in a lab environment. Other benefits are as follows:

- Location independence of the Data

- Replication independence
- Distributed-query processing
- Some level of distributed-transaction processing
- Some level of DBMS independence
- Hardware, Network, and Operating System Independence
- "Pass-Through" Capability

A cost is associated with the use of any technology. First and foremost is the penalty of not being able to use seamlessly the advanced SQL implementations that each RDBMS vendor provides without having to use "pass-through." Another disadvantage is the possibility of having to standardize on one vendor's particular access protocol and bindings. Thus, some disadvantages are:

- Rewriting application code to ensure accurate SQL-dialect translation
- Changing access protocol and bindings of non-Sybase client applications
- Possible performance bottlenecks (implications unknown in a production/live environment)
- Additional costs and maintenance of middleware products

The premise of the Navy's effort to test Sybase middleware was to enable the JMCIS-Afloat applications to access an Oracle database (instead of the native Sybase database) by employing the use of the OmniSQL Gateway in place of the native Sybase database. The goal was to prevent both the applications and users from detecting any difference in the functionality of the applications. The databases to be ported included the various intelligence databases, the Naval Information Processing Services Database (NIPSDB), and the Message Database (MSGDB).

To begin the project, all of the Sybase stored procedures were moved from the Sybase RDBMS environment into the OmniSQL Gateway, a process that was fairly straightforward. Next, all the data were loaded from Sybase RDBMS to Oracle RDBMS. This step was more complex. Special utilities had to be written to accommodate this transfer, as differences between the manners in which Oracle and Sybase handle date data types, long fields, and reserved words did not allow a simple porting of the data. When this was accomplished, the OmniSQL Gateway was configured inside the JMCIS Afloat environment and the Sybase-native JMCIS Afloat applications could be connected seamlessly to the Oracle DBMS via the OmniSQL gateway, with no difference in functionality available to the users and no code changes in the user applications. In summary, the effort was successful, with consideration to the fact that specialized utilities had to be developed to convert the data from Sybase to Oracle.

Current Access Methods, Technological Advancements, and Emerging Standards

Today in client server environments, a common problem creates a rather rigid access method for applications to communicate to data repositories. This problem is the interface from the client applications to the database. In the relational world, RDBMS vendors have developed proprietary-access methods and

bindings. Access methods to image and text files typically have been custom built (yet HTML is becoming widely used and may well become the industry's de-facto text and image standard access method).

Problems continue with current database-access methods. C^4I applications virtually always have been developed to run against a single, homogeneous database in a client-server environment using the access methodology provided by that vendor's RDBMS. This makes software-application maintenance costs higher, portability more cumbersome, and typically limits or constrains both clients and servers that are to be used for these applications. Also this situation creates a development environment where teams of software developers must be disciplined in one or more access methodologies. This is primarily a result of the different approaches that RDBMS vendors employ to deal with data access at the protocol, binding, and language levels.

At the protocol level, a proprietary method generally is layered onto the base communication protocol. Oracle uses SQL*Net, Sybase uses Open Server/Open Client, etc. Both can be encapsulated in a TCP/IP-communication layer. The problem is that each vendor has addressed this issue with a custom approach, as no standards existed when the first client-server architectures emerged in the mid-to-late 1980s.

At the binding level, the data in a database are manipulated via a language such as SQL. The problem here is that beyond the most elementary usage of SQL, each vendor has implemented their own "flavor" or dialect of SQL to deal with vendor-proprietary stored objects (such as procedures, functions, and internal-record identifiers).

At the language level, RDBMS vendors have come up with their own method of advanced, SQL-language features. Oracle's name for this is PL*SQL; Sybase calls theirs TRANSACT SQL. These language extensions, or SQL dialects do not communicate with each other above the generic SQL level. Thus, the real power of the RDBMS becomes usable only when one employs a vendor's custom SQL dialect, thus making the code less portable and tied to a particular RDBMS implementation.

In the following sections, we will examine the standards committees' progress in addressing these problems at each of these levels.

In response to the limitations and constraints dealing with data access at the protocol level, International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) have been working on SQL/RDA (Remote Data Access) [14, 15]. This standard interface defines how RDBMS servers can utilize a common-protocol interface, known as the "RDA-SQL Server Protocol Interface." SQL/RDA provides the basic services and protocols for SQL interoperability in a distributed, wide-area, client-server environment.

In response to the limitations and constraints dealing with data access at the binding level, the software industry is addressing this problem in a variety of ways. One of the most widely known is the ODBC method from Microsoft. This "common denominator" approach to translate SQL from client applications to a variety of RDBMSs works only at the lowest levels of SQL, but is effective. ANSI will be standardizing this approach in something called ANSI-SQL Call-Level Interface (CLI) [24]. The obvious drawback is the lack of a standard, such as the SQL Persistent Stored Modules (PSM), which will keep SQL/CLI operating within the same limitations of ODBC. Another reported drawback is performance penalties with this translational binding method. But at least the industry has recognized the problem and is taking steps to solve it. (See, for example [18].)

To bring some structure to the RDBMS vendors' proprietary implementation of stored objects (procedures, functions, etc.), ANSI has the SQL/PSM, mentioned above. This is the extension to SQL that provides advanced SQL capabilities in a standardized format. This extension is comprised of variables, procedures, functions, and flow-control statements that execute at the RDBMS kernel, providing robust performance and functionality. Thus, SQL/PSM will go a long way towards solving the "SQL dialect" problem between RDBMS vendors.

Another area in which ANSI has been working to solve the data-access dilemma is ANSI SQL External Repository Interface (ERI) [24]. This standard interface defines how RDBMS servers can provide limited SQL access to non-RDBMS data repositories, such as Full-Text Document Systems, Legacy Systems, Graphic Information Systems (GIS), or ODBMSs. SQL/ERI will use SQL/RDA to accommodate "calls" or "messages" to request and manipulate non-SQL data. As this type of data is increasingly being accessed simultaneously with RDBMS data, it has become necessary to provide the industry with an acceptable, common interface to accommodate this requirement. This standard will ease significantly some of the problems that developers face each time a new system is developed or expanded. This is part of SQL3 [18].

IV. A COMPARISON OF TECHNOLOGIES AND APPROACHES

All three of the major approaches, the JTF-ATD Data Server (which is CORBA compliant), DCE, and Sybase Omni-SQL Gateway provide some level of generic, distributed database access capabilities. Although DCE and CORBA provide some of the same kinds of services on networks, DCE was designed to support procedural programming in languages such as "C," whereas CORBA supports object-oriented programming in languages such as "C++." This is the main difference between the two [1]. (Another difference lies in the area of security support.) In the JMCIS environment, legacy programs in C as well as new object-oriented software will require data-access services. Therefore, the best solution for JMCIS and for other C^4I systems is to utilize both DCE and CORBA. One way to

implement this is for C programs to include calls to C++ modules when the code is complied. Some CORBA implementations can run "on top of" DCE (Brando, 1995). Other combinations of the above technologies also could prove to be useful. For example, DCE can be integrated into the products of the RDBMS vendors. (See, for example, [2].) Therefore, it is reasonable to expect that DCE also can run in the same environment as Sybase Omni SQL Gateway.

GCCS Efforts with DCE and ORB

At the time of this writing, the GCCS plan was to incorporate DCE and CORBA into the COE. GCCS supports an open-system environment and the COE is based on the distributed client-server computing model that can be implemented in many ways. At the time of this investigation, DISA had chosen the Open Group's DCE as the baseline architecture and standard for implementing a client-server computing model. DCE is an implementation designed to support environment such as that of GCCS, in which information flows across heterogeneous and distributed hardware and software platforms in a manner transparent to the end user. Future development of the GCCS's distributed environment could be object oriented with a strategy of running a CORBA-based product on top of DCE.

V. CONCLUSION AND RECOMMENDATIONS FOR FUTURE WORK

The committee's findings presented in this White Paper support the following recommendations:

- At the time of this investigation, the Global Command and Control System (GCCS) community at the Defense Information Systems Agency (DISA) was involved in a similar effort. The services in general and the Navy in particular must combine their efforts with DISA's to share findings, stay involved in the decision-making process, and remain compatible with the GCCS data-access standards.
- No single technology or method will accommodate all Naval C^4I requirements, but combining technologies in concert with the GCCS direction can accommodate nearly all requirements and facilitate the process of determining the next-generation data-access architecture for GCCS as well as JMCIS.
- Although the level of maturity of many of the of latest technologies may be insufficient to permit a full commitment, it is necessary to start working with these new technologies in order to gain insight into what works best. Thus, when some of the more advanced technology reaches a mature state C^4I systems can take full advantage of them. Personnel already will be familiar with the requirements that need to be met in order to remain interoperable with GCCS.

Several prototypes including DCE, CORBA and COTS technologies can be integrated in concert with GCCS, conforming to DISA's standards, to ensure that the

JMCIS data-access methodology would be interoperable with GCCS. This approach also would avoid duplication of effort. Thus, JMCIS could build on what already has been accomplished with GCCS as a point of departure for the next-generation, data-access services for the maritime forces.

Each of the major approaches described here needs to be further tested, including each two-way combination of each of the three subject technologies. For example, the engineer responsible for testing the JTF-ATD Data Server's implementation of CORBA also could test how DCE would fit into that environment, whereas the engineer who is testing the COTS approach could also test how Sybase OMNI would run in the same environment with CORBA and the JMCIS Federated Database.

While evaluating the prototypes, the requirements for the functionality in JMCIS should be re-analyzed and appropriate matches of technology and requirements should be fielded first at specific sites and later, for all JMCIS sites. This functionality should be developed and integrated for the JMCIS high-performance, distributed data, information base and world model according to the requirements.

A key approach to developing these advanced capabilities will be to utilize object-oriented and world-wide-web technologies, and to develop military and geopolitical classes and instances, various logical relationships and dependencies, self-updating, temporal and spatial indexing, configuration management, event monitors, and mechanisms for management of redundancy and inconsistencies due to latency, bandwidth limitation, priority, heterogeneity, and uncertainty. This effort will utilize state-of-the-art technologies to provide a federated-database management system to enable JMCIS to meet the requirements of users and applications at operational afloat and ashore command centers.

A phased approach can be implemented to identifying and validate the above the requirements, developing, and integrating the advanced information services and technologies into selected JMCIS segments. This will provide a proof-of-concept demonstration of integrating next-generation, intelligent-information collection, analysis, fusion, and dissemination services into JMCIS.

These data services will provide DoD with Object-Relational technology that results in data-source independence and provides an architecture that allows DoD to "plug and play" data-management systems.

The JTF-ATD Data Server's approach should be compared to and weighed against the approach of developing DCE and COTS tools that support application migration to object-oriented capabilities. All approaches discussed above should be explored and prototypes should be develop to assess the benefits of each approach, including the approach(s) to translating DBMS-embedded C-language calls to data services within the next-generation information infrastructure.

ACKNOWLEDGMENTS

The authors thank Perry (Peter) Cherpes, formerly of Science Applications International Corporation, for his significant contribution, and the Space and Naval Warfare Systems Command who sponsored this investigation. This article is the work of a U. S. Government employee in the capacity of official duty and is not subject to copyright. It is approved for public release with an unlimited distribution.

REFERENCES

[1] T. J. Brando, *Comparing DCE and CORBA*, MP 95B0000093, Bedford, MA: The Mitre Corporation, 1995.

[2] D. Brown, *DCE Open Client and Open Server Support of the Distributed Computing Environment*. Emeryville, CA: Sybase Corp. Technical Paper Series, 1995.

[3] M. G. Ceruti, "Development options for the Joint Maritime Command Information System (JMCIS) specialized data servers," *Proceedings of the Department of Defense Database Colloquium '96*, pp. 217-227, Aug. 1996.

[4] M. G. Ceruti "Application of knowledge-base technology for problem solving in information-systems integration," *Proceedings of the Department of Defense Database Colloquium '97*, pp. 215-234, Sep. 1997.

[5] M. G. Ceruti, "A Review of Data Base System Terminology," *Handbook of Data Management*, Chapter 1, pp.3-21, CRC Press, LLC, Boca Raton, 1998.

[6] M. G. Ceruti, M. N. Kamel, and B. M. Thuraisingham "Object-oriented technology for integrating distributed heterogeneous database systems," *Proceedings of the Department of Defense Database Colloquium '95*, pp. 79-98, Aug. 1995.

[7] P. Cooper, "GCCS gives allies unique data sharing powers," *Defense News*, October 28, 1996.

[8] R. T. Due, "Object technology essentials," *Handbook of Data Management*, Chap. 14, pp.187-203, CRC Press, LLC, Boca Raton, 1998.

[9] M. A. Farrar, "Object-oriented technology for abstracting access to object-oriented database systems and other heterogeneous sources," *Proceedings of the Department of Defense Database Colloquium '97*, pp. 299-408, Sep. 1997.

[10] M. Frank, "Database and the Internet," *DBMS Magazine*. vol. 8, no. 13, pp. 44-64, 1995.

[11] J. N. Froscher, "Security information through a replicated architecture," *Handbook of Data Management*, Chap. 13, pp. 173-183, CRC Press, LLC, Boca Raton, 1998.

[12] M. Hammer and D. McLeod, *On Database Management System Architecture*, Tech. Rep. MIT/LCS/TM-141. Cambridge, MA, Massachusetts Institute of Technology, 1979.

[13] M. Hammer and D. McLeod, "On database management system architecture," *Infotech State of the Art Report ,vol. 8: Data Design.* Pergamon Infotech Limited, 1980.

[14] International Organization for Standardization & International Electrotechnical Commission (ISO/IEC), *Information Technology - Open Systems Interconnection - Remote Database Access Part 1: Generic Model, Service and Protocol.* (ISO/IEC 9579-1, 1993(E)), Geneva, Switzerland: ISO/IEC Copyright Office, 1993.

[15] International Organization for Standardization & International Electrotechnical Commission (ISO/IEC), *Information Technology - Open Systems Interconnection - Remote Database Access Part 2: SQL Specialization.* (ISO/IEC 9579-2, 1993(E)), Geneva, Switzerland: ISO/IEC Copyright Office 1993.

[16] R. Lawson, "Developing Web-based Internet applications with reusable business components," *Proceedings of the Department of Defense Database Colloquium '96*, pp. 503-520, Aug. 1996.

[17] J. McKee, "JMCIS: 'The big picture.' " *Surface Warfare.* vol. 20, no. 4, pp. 10-11, 1995.

[18] J. Melton, "ANSI SQL3 update." *Database Programming and Design.* vol. 8, no. 11, pp. 61-63, 1995.

[19] Object Management Group (OMG) & X/Open Company Limited, *The Common Object Request Broker: Architecture and Specification*, Revision 1.1, OMG Document Number 91.12.1 Revision 1.1, 1992.

[20] P. Piper, "Defense Information Infrastructure (DII) Shared Data Environment (SHADE)," *Proceedings of the Department of Defense Database Colloquium '96*, pp. 407-418, Aug. 1996.

[21] D. C. Reilly and J. A. Flemming, "Lessons learned in legacy data access," *Proceedings of the Department of Defense Database Colloquium '95*, pp. 623-629, Aug. 1995.

[22] R. J. Richards, "Data sharing in future technology: COTS software vs. COTS objects," *Proceedings of the Department of Defense Database Colloquium '96*, pp. 573-585, Aug. 1996.

[23] A. P. Sheth and J. A. Larson, "Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases", *ACM Computing Surveys*, vol. 22, no. 3, pp. 183-236, Sep., 1990.

[24] U. S. Dept. of Commerce, National Institute of Standards & Technologies (NIST), *Federal Information Processing Standards Publication 193*. Gaithersburg, MD, 1995.

AUTHOR BIOGRAPH IES

Dr. Marion G. Ceruti
Space and Naval Warfare Systems Center
Code D4221, 153560 Hull Street San Diego, CA 92152-5001
Tel. (619) 553-4068, DSN 553-4068, Fax (619) 553-5136,
INTERNET: ceruti@spawar.navy.mil

Dr. Ceruti is a scientist in the C^4I Systems Engineering and Integration Group of the Command and Intelligence Systems Division at the Space and Naval Warfare Systems Center, San Diego (SSCSD). She received the Ph.D. degree in 1979 from the University of California at Los Angeles, where she was awarded a research fellowship from the International Business Machine Corporation. Her present professional activities include information systems research and analysis for C^4I decision-support systems. Dr. Ceruti has served on the program committee and as Government Point of Contact for all annual Database Colloquia since 1987. An active member of AFCEA and several other scientific and professional organizations, she is the author of numerous publications on various topics in science and engineering, including information management. Dr. Ceruti has received publication awards of merit and excellence from SSCSD and its predecessor.

Mr. Scott A. Gessay
FGM, Inc., 131 Elden Street, Suite 108, Herndon, VA 22070
Tel. (703) 478-9881, Fax (703) 478-9883, INTERNET: scottg@fgm.com

Mr. Gessay is a senior systems architect and the CEO of FGM, Inc. with more than 20 years of experience in large-scale software development for C^4I projects. His experience includes managing C^2 software development efforts, requirements analyses; system design; software, relational database, and user-interface design and development; systems integration, testing, training, installation, and on-site support. Mr. Gessay received his M. S. degree in electrical engineering, from the George Washington University in 1988. He has participated in the development of several major C^4I systems including the JMCIS Ashore and GCCS.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1998	Professional Paper

4. TITLE AND SUBTITLE
White Paper on the Next-Generation Data-Access Architecture for Naval C4I Systems

5. FUNDING NUMBERS
In-house

6. AUTHOR(S)
Dr. M. G. Ceruti(1), and S. A. Gessay(2)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Space and Naval Warfare Systems Center(1) San Diego, CA 92152–5001
FGM, Incorporated(2) Herndon, VA 22070

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Space and Naval Warfare Systems Command
San Diego, CA 92110-3127

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENTR
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The Joint Maritime Command Information System (JMCIS), is the primary Command, Control, Communications, Computers, and Intelligence (C4I) system for the maritime services. To promote efficient information access in this system, the JMCIS Data Engineering Group formed a committee with a three-fold mission: to evaluate the state of new data-access technology, methods and architectures; to explore the systems and software developed within the C4I sector that can use this technology; and to recommend an architecture on which to base the framework of data-access methodologies into which Naval C4I systems could evolve. The various technologies and approaches available for the next-generation data-access methodology in Naval C4I were investigated, including an examination of both commercial-off-the-shelf technologies, international standards, and general industry trends. The approach was to define the requirements, evaluate the available technologies, and to compare them to one another. The option of combining technologies was explored. The findings of this investigation are presented here.

Published in Proceedings of the Fifteenth Annual Federal Database Colloquium, "Combining Emerging Technologies for the Information Systems of the Future," pp. 451-472, September 1998.

14. SUBJECT TERMS
Mission Area: Command, Control, and Communications
Common Object Request Broker Architecture (CORBA)
Global Command and Control System (GCCS)
Joint Maritime Command Information System (JMCIS)

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	Same as Report

21a. NAME OF RESPONSIBLE INDIVIDUAL	21b. TELEPHONE *(include Area Code)*	21c. OFFICE SYMBOL
Dr. M. G. Ceruti	(619) 553-4068	Code D4221